The Bell System Technical Journal
Vol. 53, No. 8, October 1974
Printed in U.S.A.

LAMP:

Logic-Circuit Simulators

By S. G. CHAPPELL, C. H. ELMENDORF, and L. D. SCHMIDT

(Manuscript received February 20, 1974)

The algorithms used for logic-circuit simulation in the Logic Analyzer for Maintenance Planning (LAMP) system are described. Several simulators are available to allow a cost-effective tradeoff between simulation cost and the level of detail needed for a particular application. The true-value simulator provides efficient simulation of fault-free logic circuits. Two fault simulators simulate the classical stuck-at faults as well as shorted-gate-output faults. Hyperactive faults, those faults which cause an inordinate amount of simulation activity, are discussed along with their impact on simulation time. A four-value simulation logic is described which simplifies circuit initialization procedures.

I. INTRODUCTION

The use of digital simulation of logic circuits has been widely accepted in the computer and telephone industries to verify logic-circuit designs, to analyze the behavior of logic circuits in the presence of faults (such as gate outputs permanently stuck at logical 0 or logical 1, open gate inputs and shorted gate outputs), and to aid the generation of fault-detection tests for logic circuits.

Most simulators described in the literature can be divided into three categories. The first category includes the true-value simulators that simulate the circuit in the absence of any faults or, by altering the circuit description, simulate the circuit in the presence of one permanent fault.[1,2] The second category includes the parallel simulators that concurrently simulate the fault-free circuit and the effect on the circuit of a small set of single permanent faults.[2–4] The third category includes the deductive simulators that concurrently simulate the fault-

free circuit and the effect on the circuit of all single permanent faults.[5] The Logic Analyzer for Maintenance Planning (LAMP) system contains simulators from each category.

The LAMP system has been extensively used over the last four years to simulate the No. 1A and No. 4 Electronic Switching Systems to verify the logic design, to aid tbe generation of diagnostic tests, and to analyze tbe bebavior of tbe circuits in the presence of faults. Circuits containing 52,000 gates and 23,000 single faults bave been simulated using the IBM 370 Model 168 as the host machine.

The simulators in the LAMP system provide a complete range of capabilities for the design of logic circuits. Circuits and subsets of circuits can be simulated at the gate level (NAND, AND, OR NOR, NOT), at the functional level (register, memory, etc.) or at the hybrid level (a combination of gates and functions). At the gate level, gates can he modeled in sufficient detail to account for variations of such parameters as temperature and wiring capacitance. Several different classes of faults can be considered including gate outputs stuck at logical 0 or logical 1, gate inputs open, and shorted gate outputs. Facilities have been provided to help the user debug his logic design and his diagonstic tests.[6]

This paper presents a description of tbe LAMP simulators. In tbe first section, the family of simulators are described including an example of tbeir use in tbe design of a logic circuit. This is followed by a description of the common attributes of tbe LAMP simulators. In the second section, tbe basic LAMP simulator for fault-free circuits is described and is the basis for describing tbe other LAMP simulators. In the next sections, descriptions of the deductive fault simulators and functional simulators are presented. In the seventh section, the detection and elimination of a class of "hyperactive" faults is described. Finally, data on the performance of the various simulators are presented.

II. THE SIMULATOR FAMILY

This section describes the use of the various LAMP simulators during the design of a logic circuit. This is followed by a description of the common attrihutes of LAMP simulators.

As the level of logic-circuit integration increases, it hecomes more difficult to build "breadhoard" models. This often means that more empbasis must he placed on the results of logic-circuit simulation. Therefore, it is desirable to use an extremely accurate simulation model of tbe logic circuit. Unfortunately, as tbe accuracy (level of detail)

of the model increases, so does the cost of simulation. Since LAMP was designed for large circuits (up to 65,000 gates), cost is an important parameter. One way to partially circumvent this prohlem is to utilize several different simulators, each of which provides a detailed model especially tailored to optimize the simulation of a physical circuit.

2.1 Use of simulation during circuit design

Consider the design of a small processor. Given the overall specifications for the processor, the designer can create a functional level model of the circuit where the huilding hlocks include registers, memories, decoders and an arithmetic unit. Using the *functional simulator*, the design can be simulated at the functional level to verify the operation and timing of the processor. The processor can now he divided into functional units for detailed logic design of each unit. The functional units may he further divided into circuit packs containing a few hundred gates each.

The detailed logic design of the circuit pack is performed and is verified using the LAMP *true-value simulator*. The true-value simulator simulates only the fault-free circuit hy modeling the logic gates as logic elements followed hy pure time delays. This is a fast, economical simulator.

If the timing of the signals on the circuit pack is critical, the designer may wish to perform a more detailed timing analysis of his circuit using the LAMP *timing simulator*. The timing simulator[7] allows each gate to he assigned minimum and maximum time delays for the rising and falling signal transitions. The gate output is treated as unknown during the timc hetween the minimum and maximum transition delays. This provides a more detailed analysis of circuit hehavior in the presence of variations in gate time delays resulting from such factors as temperature change, gate loading, and capacitance. In addition, gate input pulses of shorter duration than the minimum transition delay are ignored and, therefore, do not affect the gate output value.

Oncc the dcsigner has verified that his logic-circuit design meets the operational spccifications, he must generate manufacturing test vcctors (circuit input stimuli) to verify the integrity of the newly manufacturcd circuit pack. Whether the designer creates the test vectors hy hand or uses the automatic-test-generation system,[8] he may use the LAMP *fault simulator* to evaluate the quality of the resulting set of input test vectors. The fault simulator simulates the effect on a

logic circuit of tbe presence of all single classical (gate input open, gate output stuck-at-zero, and gate output stuck-at-one) faults. Tbis is a deductive simulator[5] that associates witb eacb gate output a *fault list* containing those faults that will complement tbe correct (true) logic value (logical 0 or 1) of that gate. Tbe fault lists may contain any numher of faults, which theoretically allows tbe simultaneous simulation of all classical faults. Because of tbe effort required to process the fault information, the fault simulator is considerably more expensive to use than the true-value simulator. Througb the use of the fault simulator, tests can be designed, or tbe circuit can be modified, to attain the desired level of fault detection.

If the number of faults to he simulated is less than a few tbousand, it may he more economical to use the LAMP *parallel simulator* instead of the fault simulator. The parallel simulator uses parallel fault-simulation techniques[2–4] to simulate up to 2048 single classical faults in one pass. A variahle-width-fault word is utilized so that simulation time and storage are minimized. The relative merits of the parallel and deductive fault simulation techniques are presented in Ref. 9.

After the chip layout and printed-wire routing for the circuit pack is complete, the designer may choose to examine tbe effectiveness of his classical fault tests against possible shorted faults using the LAMP *shorted-fault simulator*, which simulates tbe effect on a logic circuit of tbe presence of single pairs of gate outputs sborted togetber. If two gate outputs, A and B, are shorted togetber where gate A bas tbe value logical 1 and gate B has the value logical 0, it is assumed that tbe logical 0 will dominate and the output of gate A will be forced to logical 0. A user option is available which forces logical 1 to dominate logical 0. Potential shorted faults that may be simulated include sborted adjacent pins on cbips, shorted adjacent paths on the printed wiring board, and sborted crossover points on tbe printed wiring board. Tbese data are ohtained from the manufacturing information for eacb circuit. The shorted-fault simulator uses the deductive simulation tecbnique.

After the circuit packs are designed, tbe designer can link all tbe circuit packs together to form tbe complete processor and perform the same logic verification process on tbe larger circuit with a few minor differences. The true-value and timing simulators are used botb to verify the logic design of the processor and to verify tbe diagnostic program for the processor. The various fault simulators are used to evaluate the effectiveness of the diagnostic througbout the design-cbange cycle until tbe design is complete.

2.2 Common simulator attributes

Tbe common attributes of the LAMP true-value, fault, timing, sborted-fault, parallel, and functional simulators are described below.

(*i*) The version of LAMP that is described is implemented on tbe IBM 360 Model 67 and IBM 370 Model 168 under the IBM interactive, virtual-memory operating system TSS. A version of LAMP is also available under thc IBM operating system OS.

(*ii*) The first version of LAMP (1969) contained only tbe fault simulator. New simulators have been implemented as ncedcd, and existing simulators have becn improved to producc tbe complete system for logic simulation now available in LAMP.

(*iii*) Tbe simulators can be accessed from an interactive terminal or used in the batcb mode via card input or prestored data. Interactive features include the ability to temporarily stop tbe simulation wben any specified gate changes value and tbe ability to correct from tbe terminal errors in tbe circuit design or input data.

(*iv*) Logic circuits are simulated at tbe gate level (NAND, AND, NOR, OR, and NOT) except in tbe functional simulator, whicb also accepts descriptions of higher-level blocks such as memories and registers.

(*v*) Four simulation values (0, 1, 2, and 3) are used to simulate binary-logic circuits. The simulation values 0 and 1 represent the logic values 0 and 1. Values 2 and 3 represent unknown conditions in tbe logic circuit. This is explained in more detail in Section 3.2.

(*vi*) Conditions tbat cause the output values of flip-flops to be unpredictable are detected and tbe flip-flop outputs are forced to tbe unknown statc 3 by a process called *race analysis*. Possible circuit oscillations are detected by a process called *oscillation analysis*. Botb procedures will be described in more detail in Sections 3.3 and 3.4.

(*vii*) LAMP uses discrete event simulation wbere all activity occurs at integral multiples of the basic increment of simulation time. The basic increment definition is arbitrary and may represent sucb units as nanoseconds, microseconds, or gate delays. Lists, called timing lists, are maintained by eacb simulator such that one timing list is associated witb each increment of simulation time. Each timing list contains a list of gate-

output changes scheduled to occur at that increment of simulation time. The timing list associated with the current increment of simulation time is called the current timing list.

(*viii*) Selective trace is used so that a gate output is computed only if any of the gate's input signals changed value.

(*ix*) The circuit description is contained in a set of two-way, linked-list tables, which include information ahout each gate such as the driving and driven gates, logic function, time delay, and faults to he simulated. A suhroutine, associated with each logic function, examines the gate-input values, computes the new output values, determines whether the output values have changed, and schedules the output change (if any) into some future timing list.

III. THE TRUE-VALUE SIMULATOR

The operation of the true-value simulator will he used as the hasis for the presentation of the fault simulators. A simplified flow chart of the operation of the true-value simulator is shown in Fig. 1. This

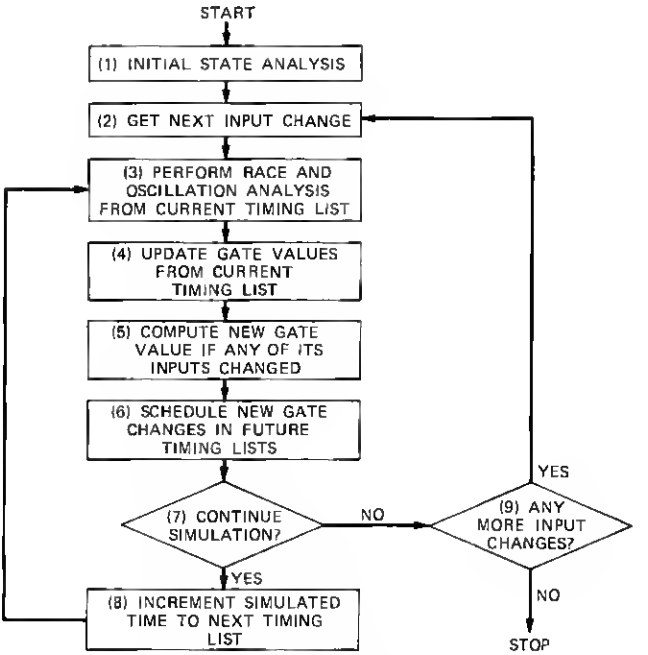

Fig. 1—Simplified simulation flow.

flow chart is also used in Section 3.5 to describe the overall simulator operation.

3.1 The true-value calculation

The LAMP simulators use four logic values, 0, 1, 2, and 3, to simulate the Boolean logic functions. The 0 and 1 are simply the logical 0 and 1 of Boolean algebra. Values 2 and 3 represent nonpropagating and propagating "don't-know" conditions, respectively. The gate output calculation occurs in Step 5 of Fig. 1.

Value 2 is used to allow efficient initialization of the circuit. Prior to a simulation run, all gates are initially assigned a value of 2. Its nonpropagating feature is demonstrated by the following table of a two-input NAND gate:

A	B	$\overline{A \cdot B}$
2	0	1
2	1	Q
2	2	Q
2	3	Q

where Q means no change in the previous true value.

The nonpropagation is necessary to prevent destroying information specified by setting *a priori* the state of the circuit. For example, in Fig. 2, if the state specification sets $C = 0$, nonpropagation is necessary to prevent the true value of C from being overwritten by a don't know. Value 2 allows $C = 0$ to initialize the flip-flop to $C = 0$ and $D = 1$. A more detailed explanation of the behavior of 2s will be presented in the next section.

True-value 3 is a true "don't know" with full propagation features. The truth table for a two-input NAND gate is shown below:

A	B	$\overline{A \cdot B}$
3	0	1
3	1	3
3	2	Q
3	3	3

where Q means no change in the present true value.

In Fig. 2, if all 2s were replaced by 3s, then the output of C and D would become 3 even though the user initialized C to logical 0.

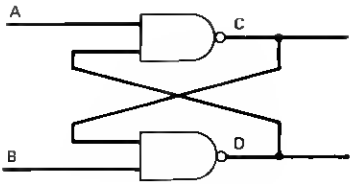

Fig. 2—NAND flip-flop.

3.2 Initial-state analysis

Tbe purposes of tbe initial-state analysis (Step 1 in Fig. 1) are: (*i*) to extract as much information as possible from the user-specified circuit state (if any), and (*ii*) to guarantee that the output of each gate is consistent witb its inputs. A flow cbart of this procedure is shown in Fig. 3.

True-value 2 is used only during the initial-state analysis which occurs before the first input vector is applied to tbe circuit. The initial-state analysis is a three-pass procedure that attempts to propagate the effect of any user-specified state through the circuit. Pass 1 has two alternatives. If the user did not set any state, tben pass 1 simply changes all of the gates whose output value is 2 to the "true" unknown-value 3 and the simulation of the input vectors hegins.

However, if the user has set some initial state, then the initial-state analysis must propagate the effect of that statc through the circuit. During pass 1, the circuit contains the logic-value 2 for the "don't-know" condition. Tbe nonpropagation feature of tbe 2s allows as much information as possihle to be extracted by the simulator using only a forward simulation. No attempt is made to set the inputs of

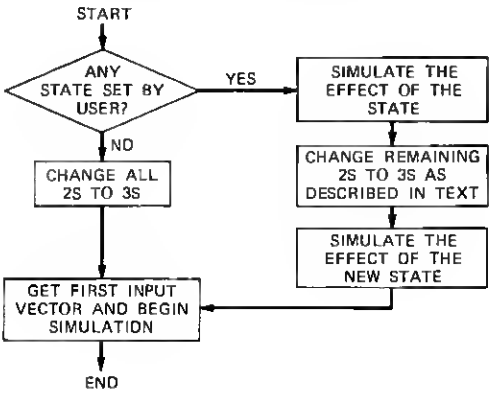

Fig. 3—Initial-state pass.

some NAND gate to logic-value 1 if the output of the gate is logical 0. Thus, LAMP requires that initial states should always be set using the "dominant" value of the particular logic type used. For example, because gate C of Fig. 2 was set to a logical 0, pass 1 would set D to value 1.

Pass 2 goes through the circuit and changes selected remaining gates whose output value is 2 to new output-value 3. This is necessary because the 3s propagate where the 2s do not. Therefore, leaving the 2s in the circuit can cause incorrect simulation results. However, it is only necessary to change to 3 those gates within maximally strongly connected subgroups (MSCs)[10] having output-value 2. This occurs because the circuit inputs are assumed to support any state which the user sets. Therefore, the input gates as well as any combinational circuitry driven by the inputs maintains true-value 2 until it is eliminated by the first input vector or the next pass.

Pass 3 propagates the newly injected 3s as far as possible. This may have the effect of destroying some incomplete state which the user specified because the circuit is unable to support the incomplete state for all possible complete states. If a complete self-supporting (stable) state is specified, no state information will be eliminated.

Initializing the circuit to some known value can introduce simulation inaccuracies during fault simulation. If the circuit is artificially initialized, there is no record of those faults whose presence would prevent the circuit from reaching the initial state specified. Therefore, it is preferable to apply a synchronizing sequence to the circuit to drive it from an unknown state (all gate output values set to 3) to some known state. The facility to artificially initialize the circuit is provided to help the user and to simplify his work.[11]

3.3 True-value race analysis

Race analysis (Step 3 in Fig. 1) is performed on the basic NAND and NOR flip-flop. Previous simulation techniques attempted to treat the flip-flop as a "black box." However, the "black box" approach leads to inaccurate simulations or to unwieldy simulation algorithms. Therefore, the technique used in LAMP is to detect races as invalid conditions on a set of gates. Since both NAND and NOR flip-flops are handled in a similar manner, only the true-value race analysis for the NAND flip-flop will be discussed here. The basic NAND flip-flop is shown in Fig. 2.

The true-value race state for a NAND flip-flop is $A = 1$, $B = 1$, $C = 1$, and $D = 1$ at the same time, t. From this state, it is impossible

to predict (assuming identical gate behavior) whether $C = 1$ and $D = 0$ or $C = 0$ and $D = 1$ when the flip-flop settles. So as not to arbitrarily resolve races, true-value 3 is assigned to the output of hoth gates in the flip-flop.

To accomplish this, when the flip-flop was in the $A = 1$, $B = 1$, $C = 1$, and $D = 1$ state at time t, the simulator calculates $C = 0$ and $D = 0$ for the new intermediate output to be scheduled into a future timing list. Since the state $C = 0$ and $D = 0$ is impossible unless the previous state was $A = 1$, $B = 1$, $C = 1$, and $D = 1$, hoth outputs at logical 0 provide an efficient race-detection mechanism.[12] Also, since $C = 0$ and $D = 0$ are unstable, both C and D will he scheduled to change values at the present increment of simulation time. Therefore, the outputs of a NAND flip-flop are set to true-value 3 and a race declared when:

(*i*) The newly calculated, but not yet assigned, outputs of both gates are simultaneously 0.
(*ii*) Both gate outputs are scheduled to be changed at the present time.

If the NAND gates are cross-coupled, as shown in Fig. 2, but are *not* specified as a flip-flop, then race analysis will not be performed. In this case, if the flip-flop is in a race state, the new output $C = 0$ and $D = 0$ will be assigned to the gates in the flip-flop. The next output (assuming the inputs to the flip-flop do not change) will be $C = 1$ and $D = 1$ and the flip-flop will oscillate hetween $C = 1$, $D = 1$, and $C = 0$, $D = 0$ causing a simulator oscillation.

In addition, hecause of the behavior of value 3, the condition where the newly calculated output values of the flip-flop are $C = 0$ and $D = 3$ or $C = 3$ and $D = 0$ will cause an oscillation. Therefore, this condition is also detected and declared to be a race. Extensive topological circuit analysis could isolate the undeclared flip-flops, but such analysis is not performed since the circuit designers seldom fail to declare the race-pair gates.

3.4 True-value oscillation analysis

A true-value oscillation (Step 3 in Fig. 1) occurs when the circuit state is unstable as a result of some input conditions. An oscillation is declared if the simulator simulates an arbitrary number, N, of increments of simulation time and the circuit has not stabilized. The value of N is defaulted to be the number of gates in the logic circuit but can be adjusted by the user.

If a true-value oscillation is detected, the old and new true values for every gate B whose output is changing at the present increment of simulated time are compared. If the old and new true values are different for gate B, the new true value is replaced with value 3 since the output of B is changing (i.e., unknown). Value 3 is the new gate output that will he scheduled in some future timing list. When 3s are inserted, the oscillation automatically stops, since a 3 represents hoth a 0 and a 1.

3.5 *The true-value circuit model*

The true-value circuit model defines the interactions among the initial-state-analysis, gate-calculation, race-analysis, and oscillation-analysis steps that were presented earlier. Thus, a description of the true-value circuit model is an overall description of the simulator operation.

A simplified flow chart of the hasic simulator operation is shown in Fig. 1. The operation includes the following:

Step 1—The circuit is analyzed to check the validity and consistency of any user-supplied initial state, as described in Section 3.1.

Step 2—This step is repeated once for every circuit input vector to he simulated. During this step, the next input vector is ohtained and the new input values are assigned to the circuit input leads. The effect of this input vector on the circuit is now propagated through the circuit. Every input gate whose value changed as a result of the new input vector is put into the appropriate future timing list. The future timing lists are examined, as the simulation time is incremented, until the first nonempty timing list is found. This timing list is called the current timing list. Let the present time he t_0 and assume that the set of gates G, $\{G_i, i = 1, 2, \cdots, m\}$, in the current timing list at t_0 contains all the gates whose outputs are changing at time t_0. Steps 3 through 6 are performed once for each timing list.

Step 3—Race analysis is performed for each declared flip-flop formed hy two gates, hoth of which are in G.

Step 4—The new outputs are assigned to every gate in G.

Step 5—After all the new outputs of G have heen assigned, the output of each gate H_j, $j = 1, 2, \cdots, n$, which is driven hy any G_i whose output has changed, is calculated according to gate model.

Step 6—If the output of some H_k, $1 \leqq k \leqq n$, changed, then H_k is put into the timing list of gates whose output may change at time $t_0 + t_t$, where t_t is the transition time for H_k. If the output of H_k did not change, no further action is taken on the gate. The important feature of this circuit model is that the gates H_j, $j = 1, 2, \cdots, n$, have their inputs calculated hased on all new values of the gates in $\{G_i, i = 1, 2, \cdots, m\}$. That is, every change that is going to occur at t_0 occurs hefore the output of any gate driven by any of the gates in G is calculated.

Step 7—Simulation may be allowed to continue or it may he interrupted to process a change on the input leads (Step 9) or to return to the command language to process user commands.

Step 8—The simulation time is incremented. This makes the timing list at time $t_0 + 1$ the current timing list and the loop continues. Simulation is terminated if there are no more input changes.

IV. THE FAULT SIMULATOR

The fault simulator utilizes Armstrong's[5] fault-list concept to allow concurrent simulation of all open gate input, output stuck-at-one (SA1), and output stuck-at-zero (SA0) faults in one pass per input vector. The input-open fault is assumed to force a nondominant value on that input. For example, for NAND and AND gates, the input open is assumed to force that gate input to logical 1. A number from 0 to $k - 1$ is associated with each of the k faults in the circuit. Each gate G, except the inverter, is assigned $N + 2$ faults, where N is the numher of inputs to gate G. The inverter has only the two output SA1 and SA0 faults, since the input-open fault is indistinguishahle from the output SA0 fault. These fault numhers are then carried in fault lists associated with each gate. The hard faults, or corresponding fault numhers, in the fault list on gate G represent exactly those faults in the circuit that will cause the true value (logical 0 or 1) of gate G to he complemented. Only gates having 1 and 0 true values can have fault lists. Similarly, the *star faults* in the fault list on gate G represent faults in the circuit for which the value of G is not predictahle by the simulation model.

4.1 Fault-simulator gate calculation

The fault-simulator gate calculation (step 5 in Fig. 1) involves the manipulation of the fault lists on each gate using the fault-list algehra.

In the description of the fault-list algebra, each fault list is treated as a set. The three set operations used for fault-list calculation are union ($\cup$), intersection ($\cap$), and difference (Θ).

The union of two fault lists A and B is defined for some fault f to form the output-fault list F:

Union Operation $A \cup B$

		B		
	F	λ	f	$^*f$
A	λ	λ	f	$^*f$
	f	f	f	f
	$^*f$	$^*f$	f	$^*f$

where

$^*f$ = star fault corresponding to fault f,
λ = absence of f and $^*f$ from the set (fault list).

The *intersection* of two fault lists, A and B, is defined for some fault f to form the output-fault list F:

Intersection Operation $A \cap B$

		B		
	F	λ	f	$^*f$
A	λ	λ	λ	λ
	f	λ	f	$^*f$
	$^*f$	λ	$^*f$	$^*f$

The *difference* of two fault lists A and B is defined for some fault f to form the output-fault list F:

Difference Operation $A \Theta B$

		B		
	F	λ	f	$^*f$
A	λ	λ	λ	λ
	f	f	λ	$^*f$
	$^*f$	$^*f$	λ	$^*f$

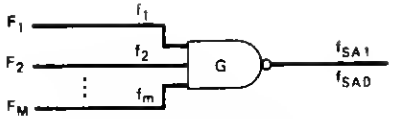

Fig. 4—Fault-list calculation.

For an m input NAND gate G in Fig. 4, let:

F_i = Fault list on the gate driving the ith input of G $(1 \leqq i \leqq m)$.
f_i = Input open on the ith input of G.
f_{SAk} = Output of G stuck at k $(k = 1, 0)$.
F = Resulting fault list on gate G.
$\overset{k}{\bigcup}$ means form the union over all the fault lists on input leads whose true value is logical k, $k = 0, 1$.
$\overset{j}{\bigcap}$ means form the intersection over all the fault lists on input leads whose true value is logical j, $j = 0, 1$.

To calculate the new output fault list F from the input lists F_i, $1 \leqq i \leqq m$, consider the following cases. First, assume all m inputs are logical 1. The output true value is 0 and

$$F = \{\overset{1}{\bigcup} (F_i \ominus \{f_i\})\} \ominus \{f_{SA0}\} \cup \{f_{SA1}\}. \tag{1}$$

This equation means that the output SA1 fault plus any fault on any input, except the respective input-open faults and the output SA0 fault, can cause the correct gate output to change values.

Second, assume that all inputs are logical 0. Then the output value is 1 and

$$F = \{\overset{0}{\bigcap} (F_i \cup \{f_i\})\} \cup \{f_{SA0}\} \ominus \{f_{SA1}\}. \tag{2}$$

This equation means that the output fault list contains the SA0 fault plus any fault present on every input lead. A fault is present on an input lead if it occurs in the lead's fault list or is the lead's input open fault. The output fault list does not contain the SA1 fault.

Third, assume that some inputs are logical 0 (those denoted by i) and the remaining inputs are 1 (those denoted by j). The output value is 1 and

$$F = \{[\overset{0}{\bigcap} (F_i \cup \{f_i\})] \ominus [\overset{1}{\bigcup} (F_j \ominus \{f_j\})]\} \cup \{f_{SA0}\} \ominus \{f_{SA1}\}. \tag{3}$$

The meaning of the equation follows directly from the meaning of eqs. (1) and (2).

Fourth, if there is a value 2 or 3 on any input and a logical 0 on some other input, then the output true value is 1 and $F = \{f_{SA0}\}$ only.

The fault-list computation equations can be derived by considering two input gates. Consider a two-input gate G with inputs A and B. If $A = B = 1$, then eq. (1) can be shown to be true by exhaustive analysis. Similarly, if $A = B = 0$, then eq. (2) is obviously true. Again if $A = 1$ and $B = 0$, then eq. (3) is true. The NAND is simply AND followed by a NOT gate and the AND operation is associative and commutative. Then eqs. (1) and (2) represent simple cascades of pairs of two input gate operations. Similarly, eq. (3) means treat all the logical 0 inputs as one AND gate G_0, then all the logical 1 inputs as an AND gate G_1, and then form the difference of G_0 and G_1. In this explanation, the internal faults were ignored. However, their handling is apparent from eqs. (1) through (3). Equations (1) through (3) describe how the LAMP fault simulator is implemented.

An alternate and more detailed implementation can be achieved by associating two fault lists with each gate whose true value is 3. The fault lists contain those faults that will cause the faulty gate output to be logical k for $k = 0, 1$. This allows more detailed analysis of faulty circuit behavior during initialization. However, this approach will significantly increase the storage required for the fault lists and the CPU time required to perform the simulation. For that reason, eqs. (1) through (3) were chosen as a realistic compromise between detail and cost.

4.2 Fault-simulator race analysis

Race analysis under fault conditions (Step 3 in Fig. 1) is performed on the basic NAND and NOR flip-flop (Fig. 2). An analogous situation to the true-value race can occur because of faults; that is, because of one or more of the faults in the fault list on gate C or D (Fig. 2). Each hard fault f in a fault list on gate G means that if f physically exists in the circuit, then the true value of G will be complemented. Therefore, the behavior of faults is identical to the behavior of true values in the faulty circuit. Then with some modification, the algorithm for detecting true-value races can also be used to detect fault-induced races. A fault f on the output gate(s) of a flip-flop (FF) is a race fault (star fault) if it satisfies all of the following conditions:

(1) Fault f will cause both outputs (D and C) of FF to be 0.
(2) Both gates of FF are scheduled to change at the present increment of simulation time.

(3) Fault f is not:

(a) Tbe input open on D from C or the input open on C from D.
(b) The output of C SA1 or SA0.
(c) The output of D SA1 or SA0.

The first two conditions are the same as the conditions for a true-value race. The third restriction is apparent since, if either of the cross-coupled inputs were open, the gates would not form a flip-flop and could not race. Similarly, either output SA1 or SA0 would make a race impossible since there is no uncertainty about the outcome. As with the true-value race, faults which force $C = 0$ and $D = 3$ or $C = 3$ and $D = 0$ will cause oscillations and are declared as raee faults.

Let F_C and F_D be the set of faults (or the fault list) on C and D, respectively. Let F_I represent the set of faults tbat cannot eause a race on FF [those faults listed in condition (3) above]. Consider three eases:

Case 1: $C = 1$ and $D = 1$; then the race faults F_R are given by: $F_R = (F_C \cap F_D) \ominus F_I$.
Case 2: $C = 1$ and $D = 0$; then the race faults F_R are given by: $F_R = (F_C \ominus F_D) \ominus F_I$.
Case 3: $C = 0$ and $D = 1$; then the race faults F_R are given by: $F_R = (F_D \ominus F_C) \ominus F_I$.

The faults in the set F_R are the star faults. These star faults are then merged into the fault list on gates C and D. That is,

$$F_C \leftarrow (F_C \ominus F_R) \cup {}^*F_R$$
$$F_D \leftarrow (F_D \ominus F_R) \cup {}^*F_R,$$

wbere F_C and F_D are the fault lists on gates C and D, and F_R is the fault list produced by race analysis. The left arrow ($\leftarrow$) means "is replaced by." Tbe new Fc and FD are assigned to gates C and D at the same time the other new output values are assigned to their gates.

4.3 Fault-simulator oscillation analysis

A fault oscillation (Step 3 in Fig. 1) is declared if the circuit does not stabilize after N increments of simulation time and no true values are changing. The number N may be set by the user as deseribed earlier.

If a fault oscillation is detected, the old and new fault lists for each gate in the set $\{G_i,\ i = 1,\ 2,\ \cdots,\ m\}$ whose inputs changed during the previous increment of simulation time are compared. Let F_{ni} = new fault list and F_{oi} = old fault list for some gate in $\{G_i\}$. Then the set of faults that are causing the fault-list changes, F_s, is determined as

$$F_s = \bigcup_{i=1}^{m} F_{si}$$

and

$$F_{si} = (F_{ni} \ominus F_{oi}) \bigcup (F_{oi} \ominus F_{ni}).$$

Since single faults are assumed, no fault can cause another fault to he in a fault list. Therefore, the set of faults that alternately appears and disappears in the fault lists must he causing the oscillation. The set of faults causing the oscillation, F_s, is flagged as star faults (or unioned as star faults) in the new list F_{ni}. That is,

$$F_{ni} \leftarrow (F_{ni} \ominus F_{si}) \bigcup {}^*F_{si}.$$

Once a true-value or fault oscillation has been detected, oscillation analysis is performed until the circuit has been stabilized. By adding star faults or adding the value 3, the circuit should eventually stabilize and the oscillation will be resolved.

Figure 5 shows a circuit that illustrates hoth true-value and fault-list oscillations. If $K1 = 1$, then the circuit will oscillate in true values. However, if $K1 = 0$, the input-open fault from $K1$ on gate $K3$ will cause the circuit to exhibit a fault-list oscillation.

Since the calculations involving the star faults are expensive, a simulator is available (logic simulator) that immediately terminates simulation of any star fault when it occurs. Thus, the logie simulator does not simulate the effeet of faults that cause "don't-know" eonditions. This approximate simulation yields faster simulation times.

V. OTHER LAMP SIMULATORS

Sections I through IV of this paper explain the fundamental ideas hehind logic-circuit simulation in LAMP. In this section, a brief description of the shorted-fault simulator and the functional simulator

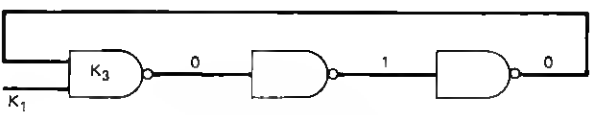

Fig. 5—Oscillating circuit.

is presented. The overall operation of the shorted-fault and functional simulators is similar to the operation of the simulators presented earlier. The fundamental difference lies in the method used to compute the output of the gate or functional element. Therefore, only the basic differences are discussed here.

5.1 The shorted-fault simulator

The shorted-fault simulator uses the deductive technique to simulate the effect on a logic circuit of a single electrical short between two gate outputs, where logical 0 is assumed to dominate logical 1. That is, if two gates, A and B, are shorted together and (in the absence of the short) A has the value 1 and B has the value 0, then in the presence of the short, gate A will have its output forced to logical 0. An option is available that causes logical 1 to dominate logical 0; however, since both cases are similar, only the case dominated by logical 0 is described here.

The shorted-fault simulator is a recent addition to the LAMP system. Because run time was expected to be considerably longer than for the fault simulator, the shorted-fault simulator was implemented to detect and immediately terminate simulation of all star faults.

The operation of the shorted-fault gate calculation requires that two fault lists, the constrained and free fault lists, be associated with each gate. The free fault list for gate A, called F_A, is computed using eqs. (1) through (3). The constrained fault list on gate A, called C_A, reflects the effects of the signals on any gates that can short to gate A. For the computation of the constrained fault lists, consider two gates, A and B, and a fault, s, whose occurrence causes the output of gate A to be shorted to the output of gate B, as shown in Fig. 6. Consider two cases:

(*i*) If $A = B = 1$,

$$C_A \leftarrow C_A \cup \{s \cap (F_A \cup F_B)\} \tag{4}$$

$$C_B \leftarrow C_B \cup \{s \cap (F_A \cup F_B)\}. \tag{5}$$

(*ii*) If $A = 1$ and $B = 0$,

$$C_A \leftarrow C_A \cup \{s \ominus (F_B \ominus F_A)\} \tag{6}$$

$$C_B \leftarrow C_B \ominus \{s \ominus (F_B \ominus F_A)\}. \tag{7}$$

The initial constrained fault list on each gate is exactly the free fault list on that gate. The constrained fault list is then altered as described in eqs. (4) through (7). These equations can be verified by examining

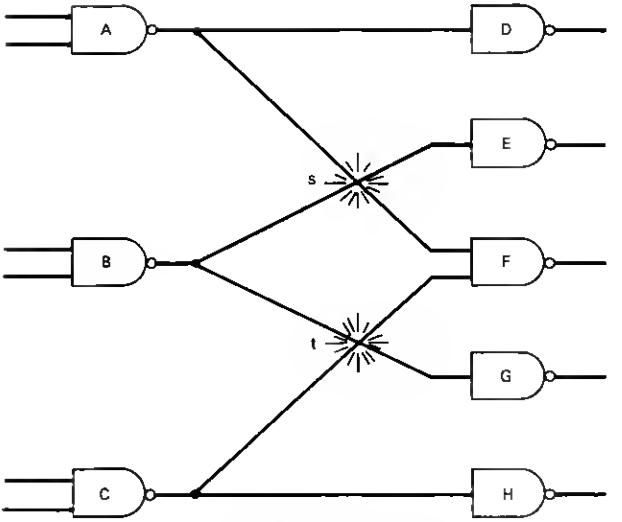

Fig. 6—Two shorted faults.

all eight cases since the only difference between the old and new constrained fault lists is fault s. This procedure must be repeated for every shorted fault that can affect the output of gates A and B (such as fault t in Fig. 6). However, since only one fault is assumed to exist at any time, all the applications of eqs. (4) through (7) are independent.

The constrained fault list on gate G is the "true" fault list for the gate since it reflects the effects of potential shorts to gate G. The free fault list on gate G is used as the starting point to compute the constrained fault list. If the free fault list were discarded after use, it would be necessary to go to the inputs of G and recompute the free fault list on G wherever it was necessary to derive a new constrained fault list on G (e.g., when a gate that could be shorted to G changes values).

The timing considerations are also important. Since the interconnecting paths are assumed to have zero time delay compared to the time delay of the gates, the effect of any shorted fault must immediately be reflected at the outputs of the gates, which may be shorted together. Therefore, the effect of possible shorts on each gate in the current timing list must be considered when the new output values are assigned to the gates (Step 4 in Fig. 1). The effect of the shorted faults may cause gates other than those in the current timing list to change value at the current time. This factor must be considered in Step 6 of Fig. 1 when the gates whose output value changed are scheduled into future timing lists.

The shorted-fault simulator has helped improve the manufacturing tests for circuit packs hy aiding the design of sets of test inputs that will detect all shorted faults.

5.2 *The functional simulator*

The functional simulator allows the simulation of higher-level functional elements, such as clocks, registers, and memories, in conjunction with gate-level simulation. Thus, the functional simulator can he used to evaluate the tentative design of a logic circuit where the entire circuit is described as functional elements. Alternatively, functional memories, registers, and clocks can he added to a gate-level simulation to provide more complete or more efficient simulation of certain hlocks hy reducing storage requirements and execution time.

The control and data flow within the functional hlock are described using an "Algol-like" language.[13] Control conditions are described using "if-then-else" statements. Data transfer is accomplished using "Assignment" statements. Such operators as NOT, AND, OR, ADD, SUBTRACT, and SHIFT are allowed. Timing information is conveyed by preceding a statement with an "at time" clause. These statements are compiled into an extended reverse Polish format[14] and executed during simulation.

The functional simulator has significantly increased the capahilities of the LAMP simulators hecause of the ease of describing a functional unit. It has heen used to aid in the logic verification of the No. 1A ESS Central Control.[11]

VI. RUN-TIME DATA

The logic and true-value simulators are the most frequently used LAMP simulators. Hence, more data are availahle on their run-time characteristics. All data shown were collected using an IBM 360, Model 67.

Table I describes ten typical circuits from a computer system. Since there is no convenient way to measure circuit complexity, two ad hoc measures are used. The numher of flip-flops in a circuit provides insight into the circuit complexity on a localized hasis while the number (or percentage) of gates in the MSCs[10] provides a more global measure of complexity. These circuits were simulated producing the data shown in Tahle II and Fig. 7.

Table II shows the simulator CPU time required to simulate the circuits described in Tahle I using the true-value, logic, and parallel simulators. The data in Fig. 7 show that the average simulator time

Table I — Size and complexity of sample circuits

Circuit	No. of Gates*	No. of Flip-Flops	No. of Gates in MSCs	Percentage of Gates in MSCs to Total Gates
A. Serial-to-Parallel Converter	349	90	224	0.64
B. Error Corrector	340	68	178	0.52
C. Parallel-to-Serial Converter	387	78	184	0.47
D. Decoder and Order Sequencer	311	15	82	0.26
E. Dial-Pulse Sequencer	336	22	112	0.33
F. Decoder and Match Circuit	383	8	44	0.12
G. Arithmetic Unit	6602	234	4378	0.66
H. Core Store Unit II	9359	320	3517	0.37
I. Core Store Unit I	2476	167	1182	0.48
J. Processor	46,012	2149	†	†

* T²L NANDs are used throughout. There are an average of two inputs per gate for the circuits listed.
† Data not available.

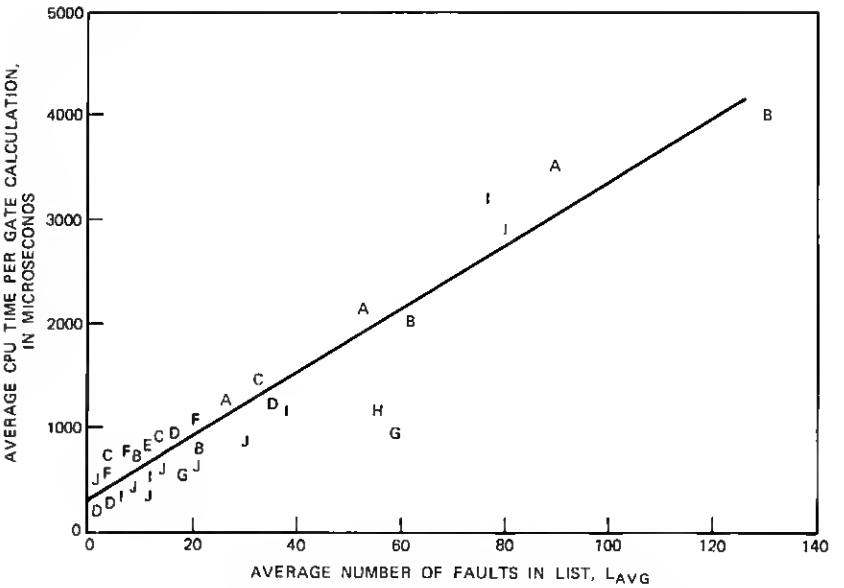

Fig. 7—Simulator time required to calculate gate output.

Table II — Simulation time for three simulators

Circuit	No. of Faults Simulated	No. of Vectors Simulated	Simulation CPU Time (Seconds)		
			Logic	True Value	Parallel
A. Serial-to-Parallel Converter	572	427	433	11	180
B. Error Corrector	894	412	641	9	102
C. Parallel-to-Serial Converter	559	348	253	9	145
D. Decoder and Order Sequencer	886	893	352	17	135
E. Dial-Pulse Sequencer	395	254	32	5	39
F. Decoder and Match Circuit	1065	161	43	3	52
G. Arithmetic Unit	2147	377	510	39	927
H. Core Store Unit II	2582	200	8361	330	*
I. Core Store Unit I	2631	16	326	17	495
J. Processor	9469	134	8673	180	*

* Data not available.

t_d required to compute the output true-value and fault list for one gate (one gate calculation) is a linear function of the length of the average fault list L_{AVG} on that gate. The length of a fault list is the number of faults in the list. The time t_d includes all bookkeeping and overhead involved in the simulation.

Figure 8 shows more data on circuit J in Table I (the No. 1A ESS processor[13]). The two lines represent the CPU time (IBM Model 67) per input vector for execution and read-write tests for the processor as a function of the number of faults being simulated. During the execution tests, the processor is executing instructions. During the read-write tests, the registers of the processor are being written and read by a second computer. The processor contains about 100,000 potential classical faults. These data were collected by simulating a subset of the faults against a subset of the diagnostic tests for tbe processor. Main memory size (4 megabytes) limits the number of faults that can reasonably be simulated, since it is desirable not to utilize the paging features of the Model 67 virtual memory because of the real-time penalty incurred due to the slow drum and disc accesses. These curves show that simulation time increases linearly with the number of faults simulated for a given set of vectors. However, the curves also show that simulation times are highly dependent on the circuit function being exercised by the input vectors.

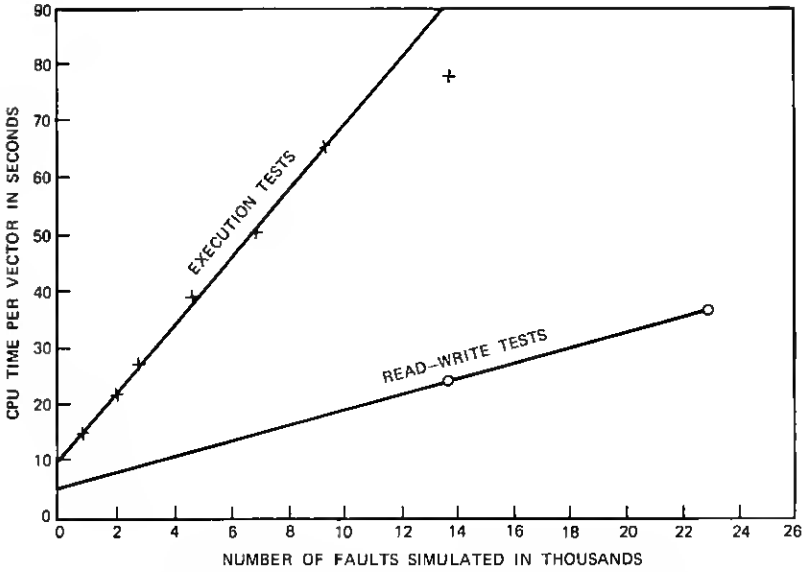

Fig. 8—Simulation times for typical processor diagnostics.

VII. HYPERACTIVE FAULTS

A new phenomenon called hyperactive faults has been found. Hyperactive faults are those faults that cause an inordinate amount of simulation activity. Removal of the hyperactive faults has reduced simulation time by as much as a factor of 8.

The fault simulator typically is more expensive to use than the logic simulator. However, it was discovered that on a 40-vector simulation of a 30,727-gate circuit with 950 faults and 152 star faults (faults which cause a race at some point during the simulation), the logic simulator took 750 seconds of IBM 360, Model 67, CPU time while the fault simulator required 2290 seconds. In an effort to determine the cause of this large discrepancy, the activity count was computed for each fault heing simulated. The activity count for a fault is incremented if that fault is in the fault list on some gate at simulated time $t + 1$, hut not in the fault list on that gate at simulated time t. The activity count is a measure of the amount of circuit activity caused by each fault.

Figure 9 shows a typical plot of the activity count distribution. For the case mentioned ahove, there were 14 faults whose activity count was more than 16 times the average activity count for all faults. These

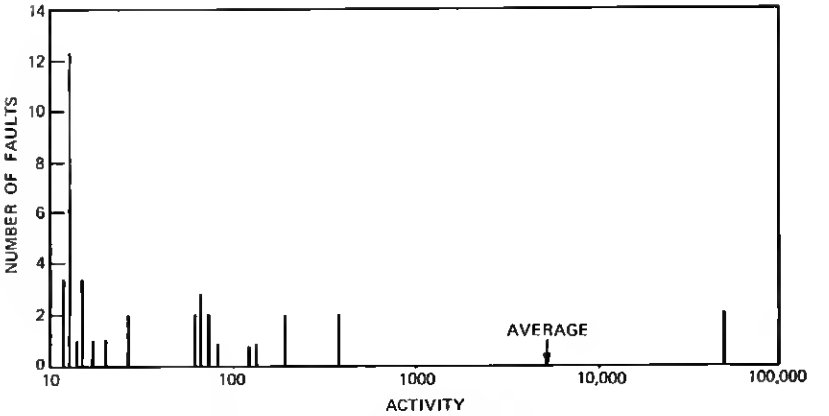

Fig. 9—Fault activity count distribution.

14 faults were removed and the circuit was resimulated with the fault simulator in 695 seconds. Thus, the removal of 1.5 percent of the faults caused approximately a 3-to-1 improvement in simulation CPU time.

Simulator speedups as high as 8 to 1 have resulted from the elimination of faults whose activity counts were excessive. For example, in a 29,696-gate circuit with 642 faults, the run time was 170 seconds per vector. By removing only 14 hyperactive faults (whose activity count was greater than 16 times the average), the run time dropped to 21 seconds per vector for the same vectors.

Two more simulations are of interest. For a 30,727-gate circuit with 1400 vectors and 2318 faults, including 601 race faults, 341 hyperactive faults were removed. On the same circuit with 3500 vectors and 5079 faults, including 1789 race faults, 101 hyperactive faults were removed. Thus, the number of hyperactive faults detected is reasonable.

Hyperactive faults are typically associated with clock circuits, sequencer circuits, and "stop circuitry." The occurrence of a hyperactive fault in the circuit often removes the effectiveness of critical control leads and causes the circuit to "run wild." While the hyperactive faults cause erratic circuit behavior, they do not necessarily cause fault-list oscillations.

The removal of hyperactive faults produces the most dramatic effect in the fault simulator because hyperactive faults are usually a subset of the star faults (race and oscillation faults) discarded by the logic simulator. Thus, the logic simulator is not as sensitive to hyper-

active faults. The removal of hyperactive faults from the fault simulator produces a significant saving in computer resources.

VIII. SUMMARY

Tbe emphasis in the LAMP simulators has been to provide a reasonable level of simulation detail in a cost-effective manner. To achieve tbis goal, several simulators bave been produced, each of whicb emphasizes some aspect of the cost-versus-detail tradeoff. The true-value simulator provides economical simulation of large logic circuits by using a true-value, integral-delay, gate-level circuit model. The timing simulator is somewhat more expensive since it analyzes minimum and maximum rise and fall delays for each gate as well as performing spike rejection in a gate-level, logic-circuit model. The logic simulator provides a two-value fault simulation using the deductive method, and a gate-level circuit model. Tbe fault simulator is identical to the logic simulator exccpt that it provides a three-value fault simulation. As a result, the fault simulator is more expensive tban the logic simulator. Clearly, both fault simulators are more expensive than the true-value simulators.

The LAMP system has been used throughout Bell Laboratories to aid logic-circuit design and analysis. LAMP, and in particular the simulators described in this paper, have been very important in the development of the ESS 1A Processor and the No. 4 ESS.[11] Because of the depth of the simulation capabilities available, LAMP has provided efficient simulation capabilities over a wide range of circuit sizes and device tecbnologies.

IX. ACKNOWLEDGMENTS

We wish to acknowledge the valuable work of G. W. Smith, Jr., R. B. Walford, and R. E. Michael on early versions of the fault simulator. We also wish to acknowledge the work of A. B. Marsb and A. M. Schowe on the functional simulator and tbe work of E. W. Thompson and D. E. Bzowy on the shorted-fault simulator. In addition, tbe encouragement and support of W. Ulrich and R. W. Ketchledge are gratefully acknowledged.

REFERENCES

1. J. S. Jephson, R. P. McQuarrie, and R. E. Vogelsberg, "A Three Value Computer Design Verification System," IBM Syst. J., *8*, No. 3 (1969), pp. 178–188.
2. S. A. Szygenda, D. M. Rouse, and E. W. Thompson, "A Model and Implementation of a Universal Time Delay Simulator for Large Digital Nets," Proc. Joint Comp. Conf., AFIPS, Spring 1970, pp. 207–216.

3. S. Seshu, "On an Improved Diagnosis Program," IEEE Trans. Elec. Comp., *EC-14*, No. 1 (February 1965), pp. 76–79.
4. F. H. Hardie and R. J. Suhocki, "Design and Use of Fault Simulation for Saturn Computer Design," IEEE Trans. Elec. Comp., *EC-16*, No. 4 (August 1967), pp. 412–429.
5. D. B. Armstrong, "A Deductive Method for Simulating Faults in Logic Circuits," IEEE Trans. on Comp., *C-21*, No. 5 (May 1972), pp. 464–471.
6. H. Y. Chang, G. W. Smith, and R. B. Walford, "LAMP: System Description," B.S.T.J., this issue, pp. 1431–1449.
7. S. G. Chappell and S. S. Yau, "Simulation of Large Asynchronous Logic Circuits Using an Amhiguous Gate Model," Proc. Joint Comp. Conf., AFIPS, Fall 1971, pp. 651–661.
8. S. G. Chappell, "LAMP: Automatic Test Generation for Asynchronous Digital Circuits," B.S.T.J., this issue, pp. 1477–1503.
9. S. G. Chappell, H. Y. Chang, C. H. Elmendorf, and L. D. Schmidt, "A Comparison of Parallel and Deductive Simulation Techniques," IEEE Trans. Comput., *C-23*, No. 11 (Nohemher 1974).
10. C. V. Ramamoorthy, "Analysis of Graphs by Connectivity Considerations," J. Assoc. Comp. Mach., *13*, No. 2 (April 1966), pp. 211–222.
11. T. G. Hallin, K. W. Johnson, and J. J. Kulzer, "LAMP: Application to Switching-System Development," B.S.T.J., this issue, pp. 1535–1555.
12. M. J. Flomenhoft, "A System of Computer Aids for Designing Logic-Circuit Tests," Proc. of SHARE/ACM/IEEE 1970 Design Automation Workshop, pp. 128–131.
13. "Revised Report on the Algorithm Language ALGOL 60," Comm. of ACM, *6*, No. 1 (January 1963), pp. 1–17.
14. D. Gries, *Compiler Construction for Digital Computers*, New York: John Wiley & Sons, 1971.